GlobalLogic®

A Hitachi Group Company

Java HandsOn Assessment

Baggage Tracking System

- Title of the Project - Baggage Tracking System
- Complexity - Medium
- Target Band - Band x / Band 1 / Band 2
- Downloadable Starter Code link:BaggageTrackingSystem
- Skills: Java, JUnit Mockito, PostgreSQL.
- Time taken to complete: 3 Hours
- IDE: IntelliJ, Eclipse

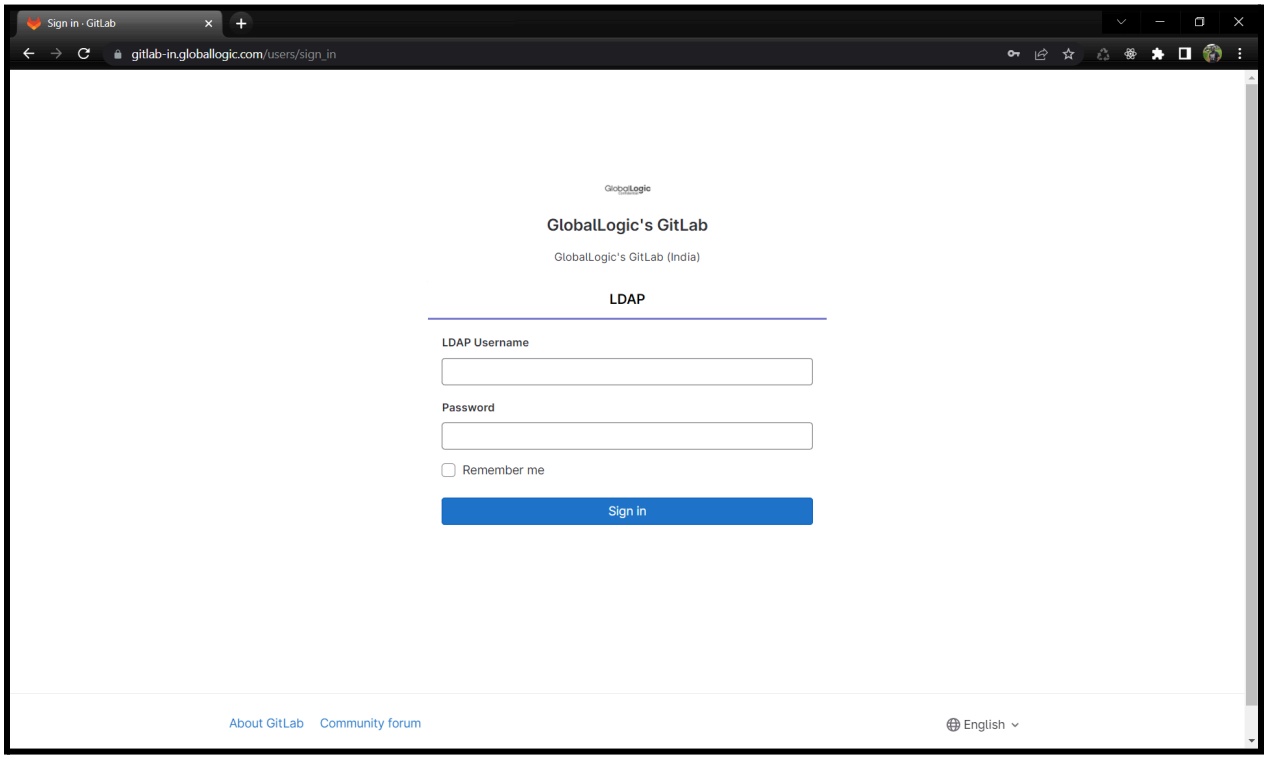
Technology	Topics
Java	OOPs, JDBC, Abstract Class and Interfaces, Exception Handling, Collections And Generics, Streams API, Lambda Expression, JUnit5 And Mockito, PostgreSQL.0

Table of Contents

1. Problem Statement 6
2. User Stories 6
3. Database Table 9
    i).Baggage 9
    ii). User 9
4.Implementation 10
    Class / Method Description 10
        Output Screens 13
5.Testing **18**
Instructions to Create and Execute the Project 18
Evaluation Metrics 20

GITLAB GENERAL INSTRUCTIONS

1. Connect to VPN to access GlobalLogic GITLAB
2. Open the URL in the browser - https://gitlab-in.globallogic.com/users/sign_in

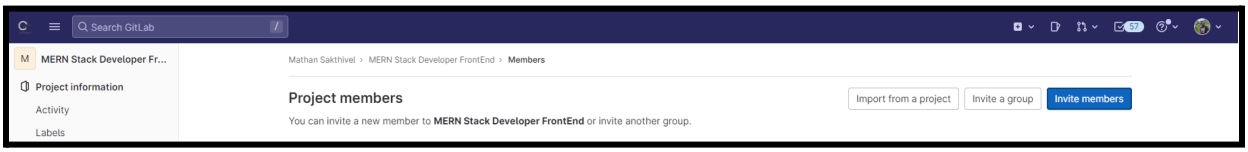


3. Login using GlobalLogic credentials.
4. Find a project to fork: Navigate to the project that you want to fork on GitLab. You can find projects by browsing the public repositories or by searching for a specific project.
5. Fork the project: Once you have found the project you want to fork, click the "Fork" button in the top right corner of the project page. This will create a copy of the project in your GitLab account.

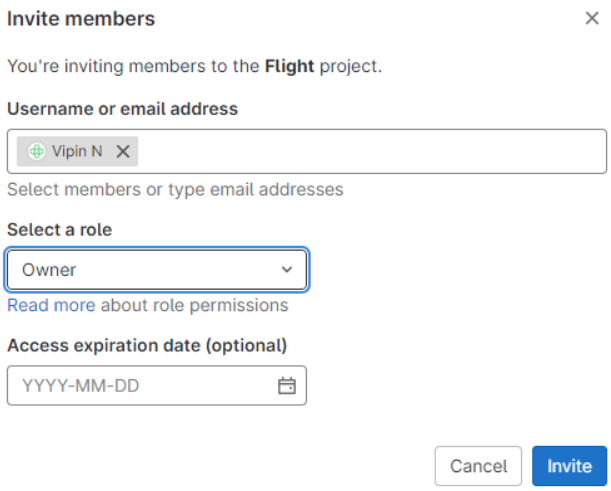
6. Clone the forked project: Once you have forked the project, clone the project to your local machine using Git. To do this, open a terminal window and navigate to the directory where you want to clone the project. Then, run the following command:
   git clone [URL of the forked project]
7. Make changes to the project: Once you have cloned the project to your local machine, make any necessary changes to the project files using your preferred code editor.
8. Commit your changes: Once you have made changes to the project files, stage the changes and commit them to your local repository using the following commands:
   git add.
   git commit -m "Your commit message here"

9. Push your changes to GitLab: Once you have committed your changes to your local repository, push the changes to GitLab using the following command:
   git push origin [name of your branch]
10. The forked project will have the Private Access so explicitly add the users in the project members.

    Click on Invite Members



Provide access to the Educators:
Username: Ex: minakshi.sharma@globallogic.com and vipin.n@globallogic.com
Role: Owner



SYSTEM REQUIREMENTS

1. Eclipse/STS/IntelliJ IDEA
2. PostgreSQL
3. Java 17 or above

Baggage Tracking System

1. Problem Statement

The Baggage Tracking System is a Java-based project designed to provide passengers with a seamless and real-time baggage tracking experience. It enables users to check in baggage to get claim tag IDs and input their baggage claim tag IDs to track the status and location of their checked baggage. The system stores comprehensive baggage details, including claim tag IDs, statuses (such as "Checked In" or "Arrived"), and locations within a relational database. Baggage handlers and airline staff facilitate real-time updates as baggage progresses through different checkpoints.

2. User Stories

User Story	Description	Acceptance Criteria
US01	As a user, I want to register an account so that I can access the system's features and services	The system should validate the input provided by the user during the registration process. This includes checking that all required fields are filled in and that the email address is in a valid format.
US02	As a user, I want to check-in my baggage so that it can be transported to my destination.	• Given that I have baggage to check-in, when I provide the location, then the system should create a unique claim tag ID for my baggage. • The system should set the status of my baggage to "Checked In". • The system should record the location of my baggage.

US03	As a user, I want to get the status and location of my checked-in baggage so that I can track its progress.	• Given that I have checked-in my baggage, when I provide the claim tag ID, then the system should return the current status and location of my baggage. • If I provide an invalid claim tag ID, the system should inform me that the baggage does not exist.
US04	As an admin, I want to view all checked-in baggage so that I can monitor the overall baggage handling process.	• When I access the system, it should provide me with a list of all checked-in baggage.

US05	As an admin, I want to update the status and location of any baggage so that I can correct any errors or update any changes	• Given a claim tag ID, when I provide a new status and location, the system should update the corresponding baggage record. • If I provide an invalid claim tag ID, the system should inform me that the baggage does not exist. • If I provide an invalid status, the system should inform me that the status is not recognized.
US06	As an admin, I want to search for a specific baggage using the claim tag ID so that I can quickly find information about a particular baggage.	• If I provide an invalid claim tag ID, the system should inform me that the baggage does not exist. • Given a claim tag ID, the system should return the corresponding baggage record.
US07	As an admin, I want to view a summary of baggage statuses so that I can get an overview of the baggage handling process.	• The system should provide a summary of how many baggage are in each status (Checked In, In Transit, Arrived, Ready for Pickup, Claimed).
US08	As an admin, I want to remove a baggage record from the system once it has been claimed so that the system stays up-to-date.	• Given a claim tag ID, when I request to remove a baggage record, the system should delete the corresponding record. • If I provide an invalid claim tag ID, the system should inform me that the baggage does not exist.

3. Database Table

i).Baggage

Column Name	Description	Data Type	Not Null
claimId	This is a unique identifier for each piece of baggage.	VARCHAR(255)	NOT NULL
location	This field represents the current location of the baggage.	VARCHAR(255)	NOT NULL
status	This field represents the current status of the baggage. The status could be values like "Checked In", "In Transit", "Arrived", "Ready for Pickup", or "Claimed". This field is also required (NOT NULL).	VARCHAR(255)	NOT NULL
userID	This field is a foreign key that links each baggage to a user in the Users table. The FOREIGN KEY constraint ensures that the userId in the Baggage table matches a userId in the Users table, maintaining the integrity of the relationship between the two tables.	VARCHAR(255)	NOT NULL
Primary Key(claimId)			
Foreign Key(userID) References User(userID)			

ii). User

Column Name	Description	Data Type	Not Null
userID	This is a unique identifier for each user	VARCHAR(255)	NOT NULL

firstName	This field represents the first name of the user.	VARCHAR(255)	NOT NULL
lastName	This field represents the last name of the user.	VARCHAR(255)	NOT NULL
email	This field represents the email address of the user.	VARCHAR(255)	Unique NOT NULL
Primary Key - userID			

4.Implementation

Class / Method Description

This is a standalone Java application where a menu-driven interface needs to be developed to execute the following functionalities.

Note:- The above UserStories are converted into equivalent methods and the descriptions of the service methods provided below need to be implemented.

Entity Class	Service Methods	Description
User(userId,firstName,lastName,email) Given User a class in the starter code convert it into Java 17 Records to store the User details.	UserService	
	registerNewUser(User user)	This method is used to register a new user.It takes a User object as input and saves it to the user table.
	checkInBaggage(Baggage baggage)	This method is used to check in a baggage. It takes a Baggage object as input, sets its status to "Checked In" and its location to "Check-in Area", and

		adds it to the baggage table.
	getBaggageInfo(String claimTagId).	This method is used to retrieve the status and location of a checked-in baggage. It accepts the claim tag ID of the baggage as an input parameter. If the baggage is found, it returns the corresponding Baggage object. If the baggage is not found, it raises a BaggageNotFoundException.
Baggage(claimId,location,status,userId)	BaggageService	
	getBaggageStatus(String claimTagId)	This method is used to get the status of a baggage. It takes the claim tag ID as input and returns the status of the corresponding baggage.If the baggage is not found, it raises a BaggageNotFoundException.
	updateBaggageStatus(String claimTagId, String status)	This method is used to update the status of baggage. It accepts the claim tag ID and the new status as input, then updates the status of

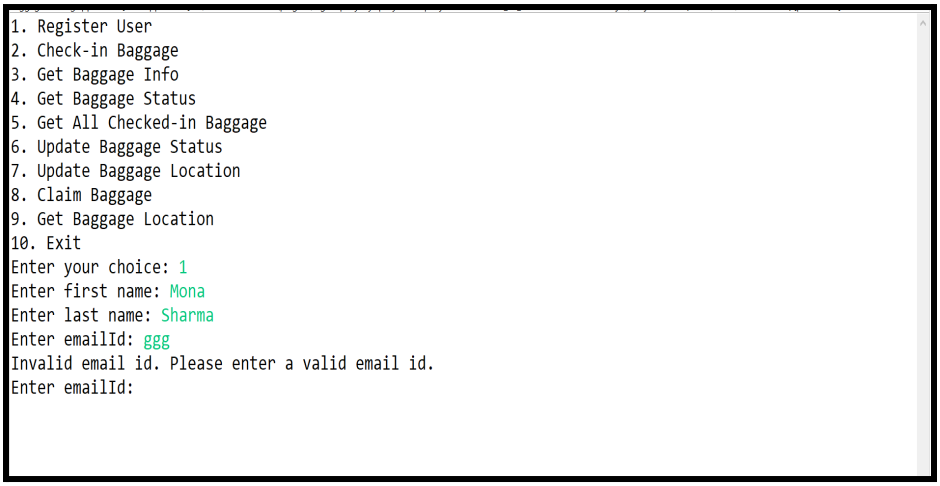
		the corresponding baggage in the database. Note: The possible status values can be "CheckedIn", "InTransit", "Loaded", "UnLoaded", "Claimed", "Lost", "Damaged". If the baggage is not found, it should throw a "BaggageNotFoundException".
	getBaggageLocation(String claimTagId) Implementation of this method is given in the starter code	This method is used to get the location of a baggage. It takes the claim tag ID as input and returns the location of the corresponding baggage else should raise BaggageNotFoundException
	updateBaggageLocation(String claimTagId, String location) Implementation of this method is given in the starter code	This method is used to update the location of a baggage. It takes the claim tag ID and the new location as input and updates the location of the corresponding baggage in the database.

	claimBaggage(String claimTagId)	This method is used to claim a baggage. It takes the claim tag ID as input and removes the corresponding baggage from the database else should raise exception.
	getAllCheckedInBaggage() Use Consumer Functional Interface and Lambda Expression to print the list of checkedInBaggages.	The getAllCheckedInBaggage method returns a list of all Baggage objects that have a status of "Checked In". else it should raise exception

Output Screens

1. Register User

i) If user enter invalid details while registration the application is giving appropriate message to the user



ii)If a user enters valid details the user is successfully stored in the database.

```
1. Register User
2. Check-in Baggage
3. Get Baggage Info
4. Get Baggage Status
5. Get All Checked-in Baggage
6. Update Baggage Status
7. Update Baggage Location
8. Claim Baggage
9. Get Baggage Location
10. Exit
Enter your choice: 1
Enter first name: Mona
Enter last name: Sharma
Enter emailId: ggg
Invalid email id. Please enter a valid email id.
Enter emailId: mona@gmail.com
User created successfully
1. Register User
2. Check-in Baggage
3. Get Baggage Info
4. Get Baggage Status
5. Get All Checked-in Baggage
```

2. Check In Baggage

```
1. Register User
2. Check-in Baggage
3. Get Baggage Info
4. Get Baggage Status
5. Get All Checked-in Baggage
6. Update Baggage Status
7. Update Baggage Location
8. Claim Baggage
9. Get Baggage Location
10. Exit
Enter your choice: 2
Enter UserID:
100
Baggage checked-in successfully
1. Register User
2. Check-in Baggage
```

3. Get Baggage Info

```
1. Register User
2. Check-in Baggage
3. Get Baggage Info
4. Get Baggage Status
5. Get All Checked-in Baggage
6. Update Baggage Status
7. Update Baggage Location
8. Claim Baggage
9. Get Baggage Location
10. Exit
Enter your choice: 3
Enter claim tag id:
1000
Claim ID: 1000
Location: Check In Area
Status: Claimed
User ID: 100
1. Register User
2. Check-in Baggage
3. Get Baggage Info
4. Get Baggage Status
5. Get All Checked-in Baggage
```

4. Get All Checked-In Baggage

```
1. Register User
2. Check-in Baggage
3. Get Baggage Info
4. Get Baggage Status
5. Get All Checked-in Baggage
6. Update Baggage Status
7. Update Baggage Location
8. Claim Baggage
9. Get Baggage Location
10. Exit
Enter your choice: 5
_____
Claim ID: 1001
Location: Check In Area
Status: Checked In
User ID: 100
_____
_____
Claim ID: 1002
Location: Check In Area
Status: Checked In
User ID: 100
```

```
Claim ID: 1003
Location: Check In Area
Status: Checked In
User ID: 23
_____
```

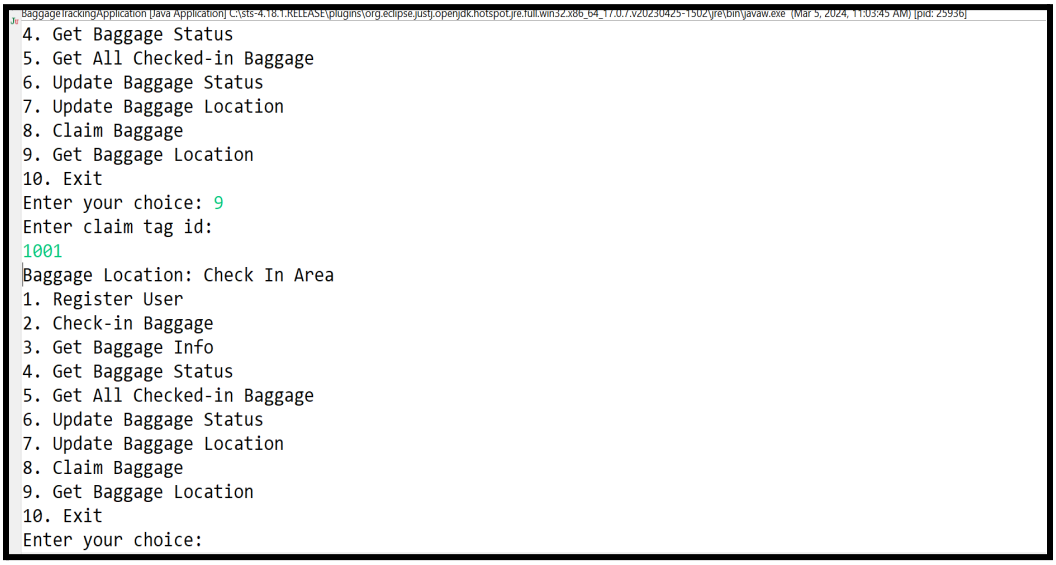
5. Update Baggage Status

```
1. Register User
2. Check-in Baggage
3. Get Baggage Info
4. Get Baggage Status
5. Get All Checked-in Baggage
6. Update Baggage Status
7. Update Baggage Location
8. Claim Baggage
9. Get Baggage Location
10. Exit
Enter your choice: 6
Enter claim tag id:
1004
Enter status:
Claimed
Baggage status updated successfully
1. Register User
2. Check-in Baggage
3. Get Baggage Info
4. Get Baggage Status
5. Get All Checked-in Baggage
```

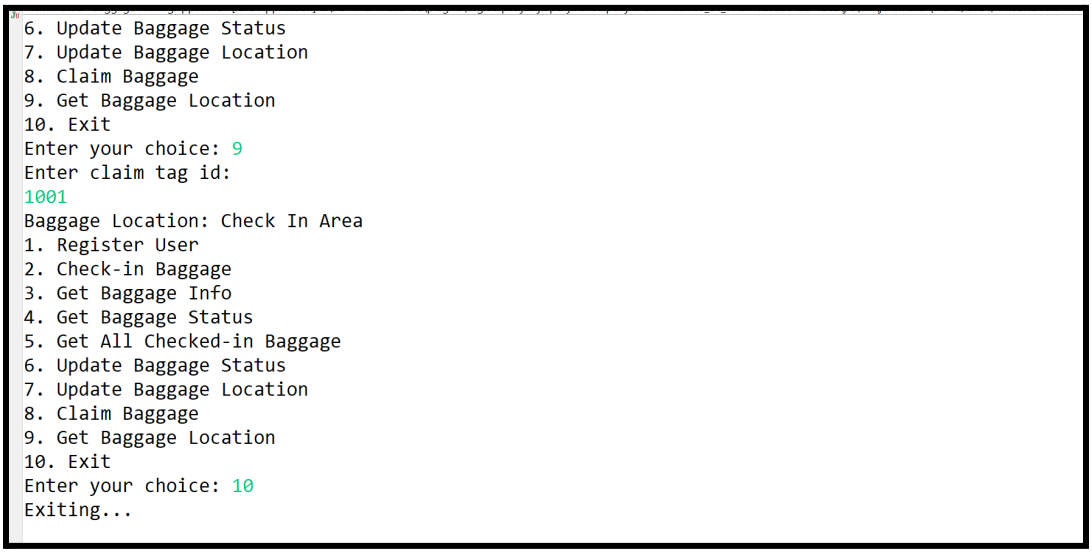
6. Claim Baggage

```
BaggageTrackingApplication [Java Application] C:\sts-4.18.1.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.7.v20230425-1502\jre\bin\javaw.exe  (Mar 5, 2024, 11:03:45 AM) [pid: 25956]
1. Register User
2. Check-in Baggage
3. Get Baggage Info
4. Get Baggage Status
5. Get All Checked-in Baggage
6. Update Baggage Status
7. Update Baggage Location
8. Claim Baggage
9. Get Baggage Location
10. Exit
Enter your choice: 8
Enter claim tag id:
1001
Baggage claimed successfully
1. Register User
2. Check-in Baggage
3. Get Baggage Info
4. Get Baggage Status
5. Get All Checked-in Baggage
6. Update Baggage Status
7. Update Baggage Location
8. Claim Baggage
```

7. Get Baggage Location



8. Exiting from application



5.Testing

Write a Test case for the getBaggageInfo () Method using Junit 5 :

- Use JUnit annotations to denote setup and test methods (@BeforeEach and @Test).
    - The setUp method initializes a Baggage instance with baggage details.
- This test method is designed to validate that the getBaggageInfo() method of the UserServiceImpl class is working as expected.
- The method takes a claimTagId as a parameter, which is the unique identifier of a baggage.
- This method retrieves the baggage info using getBaggageInfo, and asserts that the retrieved baggage matches the expected baggage.

Instructions to Create and Execute the Project

- Please download the code structure from the provided GitLab link. The project structure can be found under the 'com.gl.app' package and its subpackages.
- Design the necessary database schema to store user' information, baggage details.
- Create a Menu driven Java Project which can add Baggage and User details.
- Use PostgreSQL as the database management system.
- In addition to implementing the functionalities mentioned above(Method implementation)
    - Read Input from the keyboard using Scanner class.
    - You should include code for establishing a database connection, executing queries, and handling exceptions.
    - Use Java 8 Lambda Expressions,Streams API.
    - Field Validations(Acceptance Criteria)
        - Validate the User details(name,email).
        - claimId should have a unique value in the baggage table.
        - user_id should have unique values in the User table.
        - Establish the above-mentioned relationship between the tables to ensure the correctness of the data.

- Handle and display errors gracefully to the user in case of invalid inputs or other issues.

Evaluation Metrics

Sl. No.	Assessing Parameter	Marks
1	User Stories 8* 4 marks	32 marks
2	Usage of Lambdas and Records	5 marks
3	Testing Module	5
4	Exception Handling and following best practices	8
	Total Marks	50 Marks